



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

*Am*

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/828,049	04/06/2001	Jason Souloglou		5766

36183 7590 06/07/2005

PAUL, HASTINGS, JANOFSKY & WALKER LLP  
P.O. BOX 919092  
SAN DIEGO, CA 92191-9092

EXAMINER

CHOW, CHIH CHING

ART UNIT PAPER NUMBER

2192

DATE MAILED: 06/07/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

# Office Action Summary

Application No.

09/828,049

Applicant(s)

SOULOGLOU ET AL.

Examiner

Chih-Ching Chow

Art Unit

2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --  
Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

## Status

- 1) ☒ Responsive to communication(s) filed on 02 March 2005.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

## Disposition of Claims

- 4) ☒ Claim(s) 1-20 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☐ Claim(s) 1-20 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

## Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 02 March 2005 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.  
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).  
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

## Priority under 35 U.S.C. § 119

- 12) ☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☒ All b) ☐ Some \* c) ☐ None of:
1. ☒ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

## Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)  
Paper No(s)/Mail Date 3/17/03.
- 4) ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_.
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: \_\_\_\_\_.

### DETAILED ACTION

1. This action is responsive to amendment dated March 02, 2005.
2. Per Applicants' request, Claim 7 is amended, Claims 12-20 are amended (new), FIGs. 5-7, and Specification have been amended.
3. Claims 1-20 remain pending.

#### *Response to Amendment*

4. Applicants' Foreign Priority document filed on 03/02/2005, responding to the 08/26/2004 Office action provided in the objection of 'Priority'. The examiner has reviewed the filed Foreign Priority document respectfully.
5. The objection to the priority is hereby withdrawn in view of Applicants' Foreign Priority document, thus the effective priority date for the current application is October 10, 1998, filing date of GB Patent Application No. 9822075.9. The new effective filing date has brought to a new ground of rejection (see below).
6. Applicants' amendment dated 03/02/2005, responding to the 08/26/2004 Office action provided in the objection of drawings. The examiner has reviewed the updated drawings, Figures 5-7, respectfully.
7. The set of formal drawings filed concurrently with the above-mentioned amendment is accepted by the Examiner.

8. The Double Patenting rejections to the copending application S.N. 10/165,012 and 10/165,029 are withdrawn, since preliminary amendments canceling Claims 1-11.
9. Applicants' amendment dated 03/02/2005, responding to the 08/26/2004 Office action provided in the objection of specification. The examiner has reviewed the updated specification respectfully.
10. The objection to the specification is hereby withdrawn in view of Applicants' amendment to the specification.
11. The examiner has reviewed the newly added claims 12-20 respectfully. Claim Rejections are listed as below.

***Response to Arguments***

12. Applicants' arguments for Claims 1-4, 6-18 have been fully considered respectfully by the examiner, the Applicants' arguments are listed as following:

- Claims 1-11, "Davidson refers to memory locations of variable sizes, not variable sized registers." (page 25 of REMARKS).

Examiner's Response: Davidson teaches allocate intermediate representation into variable sized registers, for example, see Davidson 12, "Commonly-used optimizations are code motion, strength reduction, etc. Next in the internal organization of a compiler is the register and memory allocation.... At this point, further optimizations are possible, in the form of register allocation to maintain data in registers are minimize memory references; thus the program may be again rearranged to optimize register usage." 13, "In a typical compiler implementation, it is thus seen that the structure of the

intermediate language graph, and the optimization and register and memory allocation phases, are those most susceptible to being made more generic".

- Claims 1-11, "Davidson fails to teach or suggest that multiple overlapping temporary names (TN's) may be concurrently 'live'" (page 25-26 of the REMARKS).

Examiner's Response: None of the claims (1-11) have disclose this feature either.

- Claim 1-11, "Aho is assigning registers concerns physical registers, not abstract registers." (page 26 of the REMARKS) - See a prior art by Koizumi teaches 'abstract register' for intermediate representation is recited as below.

- Claims 1-11, "Neither Aho Nor Davidson teaches or suggests performing a read operation on a single variable sized register, where the read operation is effected by the data contained in the corresponding register objects".

Examiner's Response: Davidson teaches this feature in his disclosure, see Davidson 18, and 20, "The front end 20 (GEM\$XX.sub.-- COMPILE and the front-end routines that are called from it) reads source records 21 from the input stream, translates them into the intermediate representation of interface 22 (including tuples, blocks, etc. of the intermediate language graph, and the symbol table) and invokes the back end 12 to translate the intermediate representation into object code in the object file 23. ...To create an object module 23, the front end 20 translates the input stream or some subsequence thereof (which we can call a source module 21) into its internal representation for interface 22, which consists of a symbol table 30

for the module and an intermediate language graph 55 for each routine. The front end 20 then calls back end routines to initialize the object module 23, to allocate storage for the symbols in the symbol table 30 via **storage allocation (register allocation)** 28, to initialize that storage, to generate code for the routines via emitter 29, and to complete the object module 23".

- Claim 5, Lethin's filing date is 10/21/1998, which is after the effective filing date of the current application, which is 10/10/1998 (page 28 or the REMARKS). Koizumi is recited below instead of Lethin, since the certified copy of GB Patent Application No. 9822075.9 was not submitted originally.

13. Examiner is maintaining the 35 USC § 103 Rejections. For the Applicants' convenience they are listed as following.

***Claim Rejections - 35 USC § 112***

14. The following is a quotation of the first paragraph of 35 U.S.C. 112:

The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

15. Claims 12-20 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

Claim 12 recites: "The method of claim 1, wherein said variably sized registers represented by separately addressable subsets of register objects."

There is no description about "separately addressable subsets of register objects" in the specification. Claims 13-20 are rejected for the same reason as cited above.

16. Claims 14, 17, 20 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

Claim 14 recites: "The method of claim 13, wherein partially redundant expressions are concurrently attached to more than one of said separately addressable subsets of register objects."

There is no description about "partially redundant expressions" in the specification. Claims 17 and 20 are rejected for the same reason as cited above.

### ***Claim Rejections - 35 USC § 103***

17. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which

said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

18. Claims 1-4, 6-7, 10, 12-14, 18-20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Aho, Alfred et al. (hereinafter "Aho") "Compilers, principles, techniques, and tools" book as applied to claims above, and further in view of Davidson U.S. Pat. No. 5,613,117.

### CLAIMS

1. A method for generating an intermediate representation of program code written for running on programmable machine, said method comprising:

- (i) generating a plurality of register objects holding variable values be generated by the program code; and
- (ii) generating a plurality expression objects representing fixed values and/or relationships between said fixed values and said variable values according to said program code;

### Aho / Davidson

For Claim 1, first paragraph, Aho discloses the concept of generating **intermediate representation** of program code in his book, on page 12, section 'Intermediate Code Generation': "After syntax and semantic analysis, some computers generate an explicit **intermediate representation** of the source program. We can think of this **intermediate representation** as a program for an abstract machine." With respect to paragraph (i) and (ii). Aho teaches all aspects of the applicant's claims but it does not specifically mentioned "register objects holding variable values". However, Davidson shows an apparatus allows a plurality of register objects holding variable values. In Davidson, column 2, lines 39-44, "Next in the internal organization of a compiler is the register and memory allocation. Up to this point, data references were to variables and constants by name or in the abstract, without regard to where stored; now, however, data references are assigned to more concrete locations,



such as specific **registers** and memory displacements." Further in column 33, lines 41-47, "A data access tuple is a tuple which causes a **value** to be loaded from or stored into memory. (The word "memory" here includes **registers** in a register set of the target CPU 25. The only difference between a register and a normal memory location of the CPU 25 is that the 'address' of the register can only be used in a data access tuple.)"

These two paragraphs taught us that the tuples may serve as register objects. The registers are inherited from Microprocessors, in order for Aho and Davidson to use them, registers have to be generated at some point. It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement the **register allocation** of Aho with the register holding values further taught by Davidson, for the purpose of allowing the code generator to generate reasonable code for all target machines (See Davidson, column 35, lines 52-53).

For item (ii), b. an **expression object** is an operation (operator) performed for the register objects. The **expression objects** should be generated when the parsing of the program code is done. On Aho's book, page 49, under 'Abstract and Concrete Syntax' section, "A useful starting point for thinking about the translation of an input string is an *abstract syntax tree* in which each node

represents an operator and the children of the node represent the operands." Here the node can be considered as a **register object** representing a respective variables and the operator is an **expression object** that is referenced by a **register object** (operand). An example is given in Aho's book, page 558-559, each of the t2, t3, t1 and t4 are 'expression objects' and the tree shows the 'relationship' in between the expression objects, see Aho, page 290, 'Directed Acyclic Graphs for Expressions', "A directed acyclic graph for an expression identifies the common subexpressions in the expression, therefore it shows the relationship in between the expressions. Davidson further discloses this point, in Davidson, column 7, lines 43-49, "The expression may also be expressed as a logic tree as seen in FIG. 3, where the tuples are identified by the same reference numerals. This same line of source code could be expressed in assembly for a RISC type target machine, as three instructions LOAD, ADD integer, and STORE, using some register such as REG4 in the register file, in the general form seen in FIG. 3." This paragraph shows generating the intermediate representation in terms of tuples that serve as expression objects. It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement

(iii) wherein at least one variable sized register is represented by plural register objects, one register object being provided each possible size variably sized register.

the **register allocation** of Aho with the expression registers holding fixed values or relationships between the objects further taught by Davidson, for the purpose of allowing the code generator to generate reasonable code for all target machines (See Davidson, column 35, lines 52-53).

For item (iii), Aho teaches all aspects of the applicant's claims but it does not specifically mention "variable sized register is represented by plural register objects", however Davidson discloses the concept of using variable sized register. In column 72, lines 22-25, "ALLOCATE.sub.--

PERMANENT(operand, size) causes a permanent class TN (Temporary Name) of 'size' bytes to be created and referenced by the specified "operand" variable. If the "size" parameter is missing then the size of the TN is determined by the result data type of the current template."

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement the **register allocation** of Aho with the flexible sized register allocation further taught by Davidson, for the purpose of allowing the code generator sufficient flexibility to generate reasonable code for all target machines (See Davidson, column 35, lines 52-53).

2. A method according to claim 1, wherein a write operation to a variably sized register is effected by writing to the register object corresponding to the appropriate size and maintaining a record of which register objects contain valid data.

For the feature of claims 1 see Claim 1 rejection. On Aho, page 537, under 'Register and Address Descriptors', "The code generation algorithm uses descriptors to keep track of register contents and addresses for names.

1. A register descriptor keeps track of what is currently in each register.

2. An address descriptor keeps track of the location (or locations) where the current value of the name can be found at run time. The location might be a register, a stack location, a memory address, or some set of these, since when copied, a value also stays where it was. This information can be stored in the symbol table and is used to determine the accessing methods for a name." The address descriptor **keeps track of the valid data** that are stored in the register objects as described in the claim, "maintaining a record of which register objects contain valid data". Aho teaches all aspects of the applicant's claims but it does not specifically mention "writing to the register object", however Davidson discloses the concept of **reading** from and **writing** to a variable sized register. In Davidson, column 31, lines 45-50, "ALLOCATE.sub.--symbol representing a variable which is allocated to a register does not have an address, in the normal sense. However, such a symbol may be used as the address operand of a tuple which reads

from or writes to memory (a FETCH or STORE), in which case the tuple will access the indicated register."

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement the **register descriptor** of Aho with the reading and writing operations further taught by Davidson, for the purpose of allowing value to be **loaded from** or **stored into** memory. See Davidson, column 33, lines 41-47.

3. A method according to claim 2, wherein a read operation from a variably sized register is effected by determining from said record if there is valid data in more than one corresponding register object which must be combined to give the same effect as reading from the variable register, and

- (i) if it is determined that such combination is required, reading from the appropriate register object; and
- (ii) if it is determined that such combination is required, combining the contents appropriate register objects to provide a read value.

For the feature of claims 2, see Claim 2 rejection. With regard to the 'read operation from a variably sized register', please see the rejection for claim 2 (Davidson).

For item (i) and (ii), Aho discloses the skill to use multiregister for operations. See Aho, page 565, 'Multiregister Operations', "We can modify our labeling algorithm to handle operations like multiplication, division, or a function call, which normally require more than one register to perform. Simply modify step (6) of Fig. 9.23, the labeling algorithm, so label (n) is always at least the number of the registers required by the operation." The function 'label' is able to return the number of the registers required by the operation. In addition to the 'Multiregister Operation', Aho also

mentioned 'register pair' concept, page 517, under 'Register Allocation', 5<sup>th</sup> paragraph, "Certain machines require register-pairs (an even and next odd-numbered register) for some operands and results." Basically, registers can be defined in 8, 16, 32, 64 bits, it can always come in multiples. Therefore it's able to cover the function of combining many appropriate register objects needed for a certain read value. It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement the **Multiregister operation** of Aho with the reading and writing operations further taught by Davidson, for the purpose of allowing value to be **loaded from or stored into** multiple registers.

4. The method of claim 1, comprising translating the program code written for execution by a processor of a first type so that the program code may be executed by a processor of a second type, using the generated intermediate representation.

For the feature of claim 1 see rejection of claim 1. On Aho, page 463, first paragraph, "In the analysis-synthesis model of a compiler, the front end **translates** a source program into an **intermediate representation** form which the back end generates target code. Although a source program can be translated directly into the target language, some benefits of using a machine-independent intermediate form are:

1. Retargeting is facilitated; a compiler for a different machine can be created by attaching a back end of the new machine to an existing front end.

2. A machine-independent code optimizer can be done to the intermediate representation." Aho taught us that the program code from a front end processor (processor of a first type) can be translated into intermediate representation and the intermediate representation can be optimized, and then be executed to an back end processor (processor of a second type). In addition, at the beginning of the current invention spec, 2<sup>nd</sup> paragraph, the inventor also mentioned that "Intermediate representation is a term widely used in the computer industry to refer to terms of abstract computer language in which a program may be expressed, but which is not specific to, and is not intended to be directly executed on, any particular processor...". Since the 'intermediate representation' concept has been 'widely used' therefore it's not an invention.

6. The method of claim 1, comprising optimizing the program code by optimizing said generated intermediate representation.

For the feature of claim 1 see rejection of claim 1. On Aho page 463, 3<sup>rd</sup> paragraph, "A machine-independent code optimizer can be applied to the intermediate representation."

7. The method of claim 6, wherein the optimizing step is used to optimize the program code written for execution by a processor of a first pipe so that the program code may be executed more

For the feature of claim 6, see rejection of claim 6. Since the intermediate representation, which was generated from the program code that was written for execution by a processor (assuming

efficiently by that processor.

it's first 'type' instead of 'pipe'), can be optimized, therefore the program code may be executed more efficiently by that processor.

10. A system for generating intermediate representation program code written for running on a programmable machine, the system comprising:

See rejection of claim 1.

(i) means for generating a plurality of register objects for holding variable values to be generated by the program code; and

(ii) means for generating a plurality of expression objects representing fixed values and/or relationships between said fixed values and said variable values according to said program code;

(iii) wherein at least one variably sized register is represented by plural register objects, one register object being provided for each possible size of the variably sized register.

12. (new) The method of claim 1, wherein said variably sized registers represented by separately addressable subsets of register objects.

See Davidson 12, "Register allocation is also somewhat target machine dependent, and so the generic nature of the compiler must accommodate specifying the number, size and special assignments for the register set of the target CPU. Following register and memory allocation, the compiler implements the code generation phase, in which object code images are



produced, and these are of course in the target machine language or instruction set, i.e., machine-specific. Subsequently, the object code images are linked to produce executable packages, adding various run-time modules, etc., all of which is machine-specific"

13. (new) The method of claim 12, wherein said separately addressable subsets of register objects concurrently represent the same variably sized register.

Same as claim 12 rejection.

14. (new) The method of claim 13, wherein partially redundant expressions are concurrently attached to more than one of said separately addressable subsets of register objects.

Same as claim 12 rejection.

18. (new) The method of claim 10, wherein said variably sized register is represented by separately addressable subsets of register objects.

For the feature of claim 10, see rejection of claim 10. For the rest of claim 18 feature see claim 12 rejection.

19. (new) The method of claim 18, wherein said separately addressable subsets of register objects concurrently represent the same variably sized register.

For the feature of claim 18, see rejection of claim 18. For the rest of claim 19 feature see claim 12 rejection.

20. (new) The method of claim 19, wherein partially redundant expressions are concurrently attached to more than one of said separately addressable subsets of register objects.

For the feature of claim 19, see rejection of claim 19. For the rest of claim 20 feature see claim 12 rejection.

19. Claims 5, 8-9, 11, 15-17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Aho, Alfred et al. (hereinafter "Aho") "Compilers, principles, techniques, and tools" book as applied to claims above, further in view of Davidson U.S. Pat. No. 5,613,117, and further in view of U.S. Patent No. 5,586,323 by Shinobu Koizumi et al. (hereinafter "Koizumi").

**CLAIM**

5. The method of claim 4, wherein the translation is performed dynamically as the program is run.

**Aho / Davidson / Koizumi**

For the feature of claims 4 see claim 4 rejection which includes Aho and Davidson's disclosures. On Aho, page 347, under 'Static and Dynamic Checking of Types' section, "Checking done by a compiler is said to be static, while checking done when the target program runs is termed **dynamic**." Aho teaches all aspects of the applicant's claims but it does not specifically mention that the translation step is performed as the program code is run. However, Koizumi shows the concept of translating step is performed dynamically as the program code is run. In Koizumi's column 1, lines 9-14, "The present invention relates to a translation system for translating a source program into a machine language program by using an electronic computer and, more particularly, to a translation system off the type mentioned above in which an object program common to a plurality of target computers or machines of **different types can be profitably employed**. (*translation is done dynamically*)"

8. A method of generating an intermediate representation of program code expressed in terms of the instruction set of a subject processor comprising at least one variable sized register, the method comprising the computer implemented steps

(i) generating a set of associated abstract register objects representing the variable sized register;

Aho discloses the concept of generating **intermediate representation** of program code in his book, see rejection for claim 1. Aho also discloses the concept of "generating an intermediate representation of program code expressed in terms of the instruction set", in Aho's book, page 14, "we consider an intermediate form called 'three-address code,' which is like the assembly language for a machine in which every memory location can act like a register. Three-address code consists of a **sequence of instructions**, each of which has at most three operands."

With respect to item (i), Aho teaches all aspects of the applicant's claims but it does not specifically mention 'abstract register'. However, Kiozumi teaches 'abstract register' in an analogous prior art. In Koizumi column 4, lines 53-58, "an **abstract register machine** (also referred to as ARM or Arm in abbreviation) having a plurality of registers is presumed, wherein an instruction sequence for the **abstract register machine** or ARM is made use of as a basic part of the common object program (referred to as the abstract object program)".

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement the **intermediate representation** of Aho with the abstract register taught by

(ii) for each write operation of a certain field width to the variable sized register, writing an abstract register of the same width;

(iii) maintaining a record of which abstract register objects contain valid data, which record is updated upon each write operation; and

(iv) for each read operation a given field width, determining from said record whether there is valid data in more than one of said different sized abstract registers of the set which must be combined to give the same effect as the same read operation performed upon the variable size register; and

(a) if it is determined that no combination is so required, reading directly from the appropriate register; or

(b) if it is determined that data from more than one register must be so combined, combining the contents of those registers.

9. The method according to claim 4, wherein the step of determining whether or not the contents of more than one abstract register must be combined and if so which

Kiozumi, for the purpose of preserving a form of a program to be executed repeatedly (See Koizumi, column 2, lines 66-67).

With respect to item (ii), see the rejections of claim 1 (iii) and claim 2.

With respect to item (iii), see rejection of claim 2.

With respect to item (iv), see the rejections of claim 1 (iii), and claim 3; for items (a) and (b), see rejection 3 (i) and (ii).

For the feature of claim 4, see rejection of claim 4. For the rest of the claim, see the rejection of claims 1, 3, and 8.

With respect to item (i) and (ii) see rejection of claim 1 (i) and (ii), where

abstract registers must be combined, is determined in accordance with following conditions respect each set of different sized abstract registers:

- (i) if the data required for an access lies wholly within one valid abstract register, that register only is accessed; and
- (ii) if the data required for an access lies within more than one valid abstract register, data is combined from those valid abstract registers to perform the access.

11. A system for generating an intermediate representation of program code expressed in terms of the instruction set of a subject processor comprising of at least one variably sized register, the system comprising:

(i) means for generating set of associated abstract register objects representing the variably sized register;

(ii) means for writing, for each write operation of a certain field width to the variable sized register to an abstract register object of the same width;

(iii) means for maintaining a record of which abstract register objects contain valid data, the record being updated upon each write operation; and

(vi) means for determining from said record, for each read operation of a

recited that "A data access tuple is a tuple which causes a value to be loaded from or stored into memory. (The word "memory" here includes **regist**)". Only the required size is allocated, if data lies wholly within one valid abstract register, that register only is accessed. If the data required for an access lies within more than one register, that much amount of register space would be allocated (i.e. multiple registers may be combined).

See rejection of claim 8. For item (a) and (b) see rejections of claim 3 (i) Multiregister Operation, and claim 9 (i) and (ii).

Art Unit: 2192

given width, whether there is valid data in more than one said different sized abstract registers of the set which must be combined to give the same effect as the same read operation performed upon the variable size register, and

(a) if it is determined that no combination is so required, reading directly from the appropriate register; or

(b) if it is determined that data from more than one register must be so combined, combining the contents of those registers.

15. (new) The method of claim 8, wherein said variably sized register is represented by separately addressable subsets of abstract register objects.

For the feature of claim 8, see rejection of claim 8. For the rest of claim 15 feature see claim 12 rejection.

16. (new) The method of claim 15, wherein said separately addressable subsets of abstract register objects concurrently represent the same variably sized register.

For the feature of claim 15, see rejection of claim 15. For the rest of claim 16 feature see claim 12 rejection.

17. (new) The method of claim 16, wherein partially redundant expressions are concurrently attached to more than one of said separately addressable subsets of abstract register objects.

For the feature of claim 16, see rejection of claim 16. For the rest of claim 17 feature see claim 12 rejection.

*Conclusion*

20. The following summarizes the status of all the claims:

35 USC § 112 (1<sup>st</sup> paragraph) rejections: Claims 12 - 20

35 USC § 103 rejections: Claims 1 - 20

21. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

**Lethin** U.S. Patent No. 6,463,582, discloses a method for optimizing object code translation system.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chih-Ching Chow whose telephone number is 571-272-3693. The examiner can normally be reached on 7:30am - 4:00pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306. Any inquiry of a general nature of relating to the status of this application should be directed to the **TC2100 Group receptionist: 571-272-2100**.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair->

Art Unit: 2192

direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Chih-Ching Chow

Examiner

Art Unit 2192

May 28, 2005

CC

  
WEI Y. ZHEN  
PRIMARY EXAMINER